接口

typescript 的核心原则之一是对值所具有的结构进行类型检查. 它有时被称做 '鸭子辩型化'

在typescript里, 接口的作用就是为这些类型命名和为你的代码或第三方代码定义契约

```
interface LabelledValue {
  label: string;
}

function printLabel(labelledObj: LabelledValue) {
  console.log(labelledObj.label);
}

let myObj = {size: 10, label: "Size 10 Object"};
printLabel(myObj); // Size 10 Object
```

可选属性

带有可选属性的接口与普通的接口定义差不多，只是在可选属性名字定义的后面加一个`?`符号。

可选属性的好处之一是可以对可能存在的属性进行预定义，好处之二是可以捕获引用了不存在的属性时的错误。 比如，我们故意将 `createSquare` 里的 `color` 属性名拼错，就会得到一个错误提示

```
interface SquareConfig {
  color?: string;
  width?: number;
}

function createSquare(config: SquareConfig): { color: string; area: number } {
  let newSquare = {color: "white", area: 100};
  if (config.clor) {
    // Error: Property 'clor' does not exist on type 'SquareConfig'
    newSquare.color = config.clor;
  }
  if (config.width) {
    newSquare.area = config.width * config.width;
  }
  return newSquare;
}

let mySquare = createSquare({color: "black"});
```

只读属性

一些对象属性只能在对象刚刚创建的时候修改其值。 你可以在属性名前用 `readonly` 来指定只读属性

TypeScript具有 `ReadonlyArray<T>` 类型，它与 `Array<T>` 相似，只是把所有可变方法去掉了，因此可以确保数组创建后再也不能被修改

```
interface Point {
    readonly x: number;
    readonly y: number;
}

let a: number[] = [1, 2, 3, 4];
let ro: ReadonlyArray<number> = a;
ro[0] = 12; // error!
ro.push(5); // error!
ro.length = 100; // error!
a = ro; // error
```

额外的属性检查

对象字面量会被特殊对待而且会经过 *额外属性检查*，当将它们赋值给变量或作为参数传递的时候。 如果一个对象字面量存在任何“目标类型”不包含的属性时，你会得到一个错误

函数类型

接口能够描述JavaScript中对象拥有的各种各样的外形。 除了描述带有属性的普通对象外，接口也可以描述函数类型

为了使用接口表示函数类型，我们需要给接口定义一个调用签名。 它就像是一个只有参数列表和返回值类型的函数定义。参数列表里的每个参数都需要名字和类型

```
interface SearchFunc {
  (source: string, subString: string): boolean;
}

let mySearch: SearchFunc;
mySearch = function(source: string, subString: string) {
  let result = source.search(subString);
  return result > -1;
}
```

可索引类型

与使用接口描述函数类型差不多，我们也可以描述那些能够“通过索引得到”的类型，比如`a[10]`或`ageMap["daniel"]`。 可索引类型具有一个 *索引签名*，它描述了对象索引的类型，还有相应的索引返回值类型

```
interface StringArray {
  [index: number]: string;
}

let myArray: StringArray;
myArray = ["Bob", "Fred"];

let myStr: string = myArray[0];
```

类实现接口

```ts
interface ClockInterface {
    currentTime: Date;
}

class Clock implements ClockInterface {
    currentTime: Date;
    constructor(h: number, m: number) { }
}
```